Anil Vaitla & Neel Patel

Introduction:

There is a lot of interesting work currently being done in human computer interaction, both in hardware and software. As useful as the mouse and keyboard are, they can often feel unnatural when it comes to tasks such as drawing, doing 3D modeling, or playing video games. Our goal in this project was to write a gestural user interface for the game Little Fighter 2 (See Fig. 1). Little Fighter 2 is multiplayer game where players can move their characters left, right, up, down, and perform a defend motion, jump motion, or attack motion. In addition, if a certain key combination is pressed in a specific order, such as defend, up, attack, a certain more powerful action, such as throwing a large fireball, takes place. In many ways this is unintuitive. It would be more natural if a player could simply say the word "fireball" or form a fist gesture and quickly extend their fingers outwards and have the fireball emerge on screen. In addition, certain moves in the game can only occur given a specific orientation of the character. These subtleties need to be taken into account in order to create a system that provides a fluid interface for the user. For these reasons we felt that Little Fighter 2 would be a great platform for demonstrating how successful gestural and audio interaction systems can really be for nontrivial interaction. Our goal in this project is to be able to play Little Fighter 2 naturally using audio and physical gestures.

The Demo Scenario:

To demonstrate the functionality of our project, we allowed two volunteers to play LF2 against each other in a live presentation. We had one computer running Windows 7, the 3Gears API, Little Fighter 2, and our software with an external microphone and two vertically-mounted Kinects. Both volunteers used the same computer simultaneously and shared a monitor and keyboard to play the game. We also had a speaker connected to the system in order to aid in calibrating the audio recognition system. One volunteer used the gestural interface as well as a control mapping that used the left side of the keyboard. The other volunteer used the audio recognition system and a control mapping on the right side of the keyboard. We were able to train the audio recognition system to recognize three separate commands at higher than 90% accuracy in under five minutes. We had already preconfigured the server to map particular combination attacks to these keywords, but, even without a GUI to do so, we would be able to configure the server to recognize an arbitrary keyword or even an entirely unfamiliar combination attack within two minutes. The static gesture interface was used, but we felt confident enough in its accuracy that we did not demo training. It uses the same underlying server as the audio system to perform attacks. We were able to teach our gestural interface volunteer to use the system within two minutes. As a result of our system, we were able to train two users to play Little Fighter 2 at a significantly higher level with less than six minutes of total training time than our experience would indicate possible by explaining the necessary controls and key sequences required to perform combination attacks in the same amount of time. We believe this to be a significant demonstration of the ease-of-use and practicality of gestural and voice interfaces.

Our Goals:

Our project originally aimed for the following timeline:

1. Implement hand reconstruction and finger-tracking using 3Gears Systems API (Completed)
2. Write an interface that allows a Java program to emulate keyboard input (Completed)
3. Programmatically detect stationary gestures using collected finger-tracking and hand-recognition data (1 week. Expected 2/20)
4. Combine gesture recognition with keyboard emulation interface and calibrate until adequate performance[1] is achieved (1 week. Expected 2/22)
5. Expand functionality to allow non-stationary gestures to produce sequences of keystrokes maintaining level of performance (expected: 3/1)
6. Allow customization of emulated keystrokes. Do not attempt customizable recording of gestures (note: this deviation from timeline is a consequence of perceived difficulty and lack of significant existing work in the field) (expected: 3/1)
7. Ensure consistent performance across a versatile set of tasks (expected: 3/3)
8. Attempt to detect vocal commands and map these commands to keystroke interface (expected: 3/10, may be significantly more challenging than expected)
9. Integrate this vocal detection with existing keystroke interface with strong focus on performance (expected: 3/10, assuming previous checkpoint is met)
10. Develop interface for emulating mouse clicks and scroll-wheel (this and future checkpoints are expected post-deadline)
11. Map mouse interface to existing input components
12. Ensure fluid performance across versatile set of use cases
13. Integrate off-the-shelf speech-to-text functionality
14. Develop interface for emulating mouse movements
15. Map finger-tracking to this emulator
16. Smooth performance and integrate all components into a GUI control panel
17. Allow recording of customized gestures
18. Finalize product with polished user-profiling functionality
19. Develop voice-to-navigation technology

As expected, we were able to accomplish tasks 1 through 9. Our "Minority Report Demo" makes significant progress on 14 and 15, and our work with HMMs allows functionality of 17. Finally, as a consequence of design, we have relatively strong user-profiling. Therefore, with short work done on creating a GUI control panel as well as further calibration and implementation of HMMs and mouse control, we believe we could have a product that attempts all steps except 13 and 19. Since our product is not well suited to untrained audio classification, we do not believe that we will ever be able to accomplish speech-to-text functionality of 13 using our existing work. However, this problem has been solved with varying degrees of success with software such as Sphinx and Google Voice API. We would like to return to it if and when we think we would be able to add some value to the current solutions. In that vein, we believe that UIs will be developed with input devices beyond the traditional keyboard and mouse in mind. As this occurs, we think that our software will become increasingly relevant as an input device and we think that apps that can be fully navigated using gestures and voice are in the near future.

Nevertheless, we believe that our initial goal of creating an input interface that used voice and gestural commands that would allow users to play Little Fighter 2 more naturally than was previously possible has been a success.

Project Architecture:

Hardware Inputs

We use a setup produced by the 3Gears team for hand reconstruction. The physical hardware is a stand with two vertically-mounted Microsoft Kinects. Their SDK provides (x,y,z) coordinates as well as quaternions representing the angle and axis of rotation for all joints in both hands. The setup is shown in Figure 1. The audio recognition setup only requires an embedded or external microphone and speakers to listen to and confirm training set recordings.

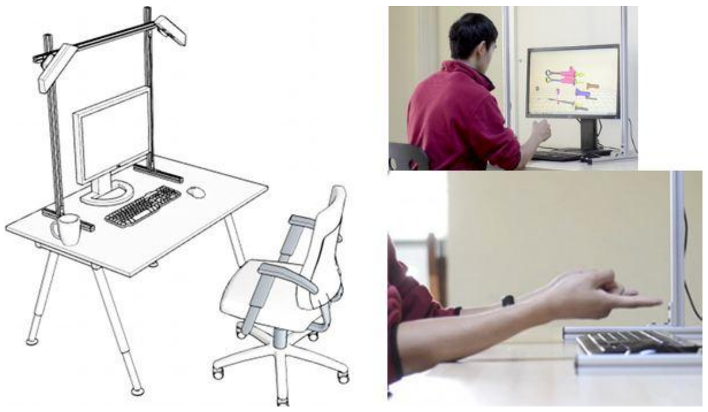

Figure 1: Example of the 3Gears Setup

Architecture Overview

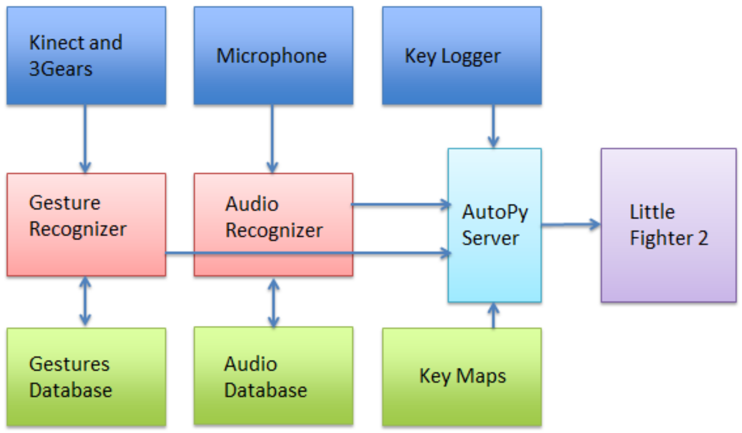

Figure 2: General Overview of the Architecture

Figure 2 outlines the general design of the system. The audio recognition and gesture recognition system receive inputs from the microphone and Kinect respectively. During real time classification they query for the trained models in the databases, and once they classify a gesture they send a message consisting of the username and gesture name to the AutoPy Server. All usernames and gesture names are resolved to the correct key combination through the keymap database and config files. AutoPy initiates the corresponding system key-up and key-down events, with a calibrated timing that allows the move to be recognized as a combination by LF2. A keylogger also sends messages to the AutoPy server to maintain a record of the current game state, allowing us to correct identify the direction of a particular combination attack.

A**udio Recognition Methodology:**

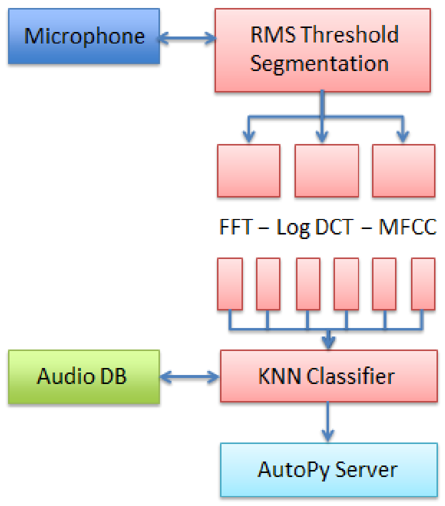

Figure 3: Outline of Audio Recognition

The user begins by recording themselves saying a single audio command such as “pow” multiple times. The system takes this single audio stream and segments it into the multiple samples. Audio onset and silence is detected by comparing a background noise threshold with the root mean square value of the audio frame buffer. Each detected sound sample gets sliced into 30ms components, and 13 MFCC components are extracted from each of these samples. So, if the sound “pow” was 900ms long, then our feature vector has 30 components, each of which has 13 MFCC features. For classification, we proceed by doing segmentation on the real-time stream. Once an utterance, or period of high amplitude audio, is complete, we split the capture into 30ms chunks and extract MFCC features. We then examine which audio samples in our training set have same length within ±30ms of the recorded audio, and find the single nearest neighbor to classify this new audio sample as. The metric used in nearest neighbor processing is L1 norm of all components in the MFCC feature vector. We tested numerous other metrics and classification criteria, such as classifying by lowest average distance across the training set as well as variations of both of these methods with outliers removed. We feel that an L1 norm provides adequate suppression of outlying features and over similar outliers within a training set are incredibly rare, allowing single nearest neighbor to be more effective than averaging methods.

Static Gesture Recognition Methodology:

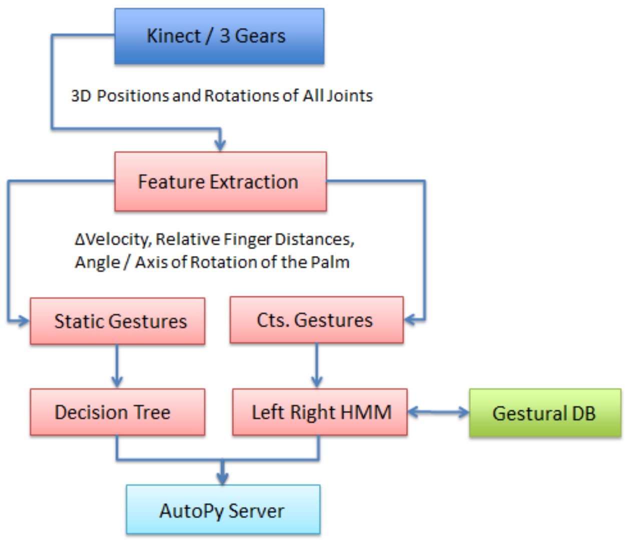
Kinect / 3 Gears
3D Positions and Rotations of All Joints
Feature Extraction
ΔVelocity, Relative Finger Distances,
Angle / Axis of Rotation of the Palm
Static Gestures
Cts. Gestures
Decision Tree
Left Right HMM
Gestural DB
AutoPy Server

Figure 4: Outline of Gestural Recognition

Static Gestures are found by looking at a subset of primitive features, specifically, the velocity vector of the hand and the pointer finger, whether the hand is a fist or outstretched (computed by checking finger tip distances), whether both hands participate in the action, and the orientation of the hand found from looking at the rotation of the palm. These three features were enough to separate our three primary static gestures “right punch”, “hadouken”, and “hand explode”. Although these require motion and don’t seem static, we label them as such since we can perform classification by only considering a single frame of data using the velocity information of that frame.

Continuous Gesture Recognition Methodology:

Continuous gesture recognition involves being able to detect some sort of movement that cannot be identified by looking at a single frame in isolation. Drawing shapes, letters, and sign language all fall into this category and cannot be identified with the static recognition system.

Continuous gesture recognition begins with the user recording multiple samples of a single continuous gesture. Then from the entire set of motions, the user must choose which frames to persist and which ones to remove. In this regard, a single continuous gesture is defined as a collection of frames. For each gesture, an HMM Model is trained with a Baum-Welch algorithm and seeded with a Left-Right transition matrix. The seed matrix for all gestures looks as follows:

.5	.5			
	.5	.5		
		.5	.5	
			.5	.5
				1

All empty entries denote 0's. Element A_ij denotes the probability of transitioning from state i to state j. In the HMM view, the hidden states of the HMM model correspond to a portion of a gesture. For example, the gesture for the number 7 seems to have two primary hidden states, one when you move your finger horizontally, and another when you move your finger diagonally. The transition matrix is the probability of transitioning from any hidden state to another, and has a strictly upper diagonal structure, in order to model the transition from one state to another. This is why this HMM System is commonly referred to as a Left-Right HMM, because you move in a linear motion between states. In our system, all HMMs are currently fixed with 5 hidden states during training. Finally the emissions, or the observations, are discrete code points denoting direction of motion of the hand. In order to compute these code points, we find the change in position of the hand, and map this direction onto the 2D plane and find the angle it is closest to as seen in Figure 3. Our emission probabilities for the 12 code points are all initialized at 1/12. Our start probability is 1 for state 1 and 0 for all other states.

Once we have built an HMM model for each set of continuous gestures, we proceed to do classification. We record the code points of the hand, and use the Forwards-Backwards algorithm[1] in order to determine which state we believe the users hand to be in. If any gesture is signaled as being in State 5[2] we indicate that the user has made the gesture corresponding to the HMM which was triggered. Currently our system can detect Symbols 6 and 7, with 30 training samples provided for each.

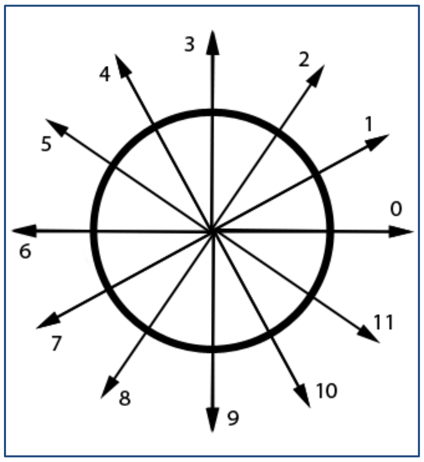

Figure 5: Codepoints for Continuous Recognition System

[1] Here the backwards variable is set to 1 because we are running in real-time

[2] This means historically the user moved from state 1 to 2 to 3 to 4 to 5 because of the structure of the transition matrix

Results:

We set up automated tests for our audio recognition performance in which the training data was randomly split into a new test and train set. Given the words “play”, “cannon”, and “shoot”, it had 100% accuracy on “play” and “cannon”, 90% accuracy on “shoot”, and only classified 3 / 20 false positives. Nevertheless, we noticed our system performance would degrade over time because a user would say a word differently then how they had recorded it. For gestures, we didn’t have automated tests, but ran a trial run in which we counted the numbers of correct and incorrect classifications. All three static gestures were recognized 100% of the time, however there were some gestures which were not among the set which were also recognized. Our continuous gesture system had poor performance in the opposite regard. When it did make an classification it would be correct, but there were many instances of 6’s and 7’s which it failed to label. The overall success rate of the continuous system was about 60%, in that it assigned no class label to 4/10 entries. This might be improved by adding more training samples or changing the number of hidden states for the underlying model.

Potential Improvements:

In doing this project there were a number of different solutions we considered and evaluated. Initially we made attempts to use the CMU Sphinx project or the Google HTML5 Audio System, but in both cases we found that although they can do untrained classification, we often want to perform training in case the commands we issue cannot be registered by the voice systems (such as the difference between “how” and “cow”) or if we want to use non-english words. Furthermore these systems are doing things more complex then we needed. They do voice to text, whereas we only required voice recognition. Therefore after doing some quick prototyping we realized our system was simple, fast, and easy to use for this project and we decided to stick with it as the basis for audio recognition. Our classifier ended up implementing a nearest neighbor metric, although one alternative we also researched is the method of dynamic time warp, or finding the optimal alignment between two audio samples before computing their distance. We implemented this method, and found that it was cross labeling audio samples too frequently, and thus we decided against using with it. In addition there are a few more interesting audio features we may consider looking into in the future. We may also try other classification methods such as using HMM or standard off the shelf classifiers like SVM. Another interesting algorithmic problem we can pursue is how to do online learning to continuously refresh our training set, because we noticed that over time there are significant deviations in the way we say words.

There were a number of design decisions to be made on the side of gesture recognition as well. One big issue was whether we could even proceed to use 3Gears since at the start of the project we weren’t even familiar with what their system could do, and if it would be suitable for our task. We also wondered whether we should switch to methods using 2D cameras. Once we felt comfortable with the available functionality from the 3Gears systems, the next issue was what classification methods to use. For static gestures, it was clear from the beginning that decision trees might work and they did perform well. On the other hand true continuous gestures were much more difficult, even though there are many libraries for Kinect developers to use. The first main algorithm is Dynamic Time Warping, which we didn’t investigate in the context of gestures fully, and the other is HMM which has a very rich history with things like sign language recognition. We chose the route of HMM, because we found it to be better documented in the literature, and we could extend the research approaches which used 2D cameras to work in our setup fairly easily. One of the main issues with the HMM approach is the need for a large amount of training samples, as well as a good understanding of the underlying transition matrix in order to avoid nasty numerical bugs. In terms of further extensions, one issue with HMM is states are dependent only on the preceding state, whereas conditioning on events further back may be even more fruitful. The next task will be to implement more powerful graphical models such as conditional random fields.

Reflections on Learning Experience:

In addition to learning a great deal about machine-learning, classification problems, and the 3Gears API, we also learned some more general lessons about software development. We learned that there is a significant cost, in terms of time, associated with over-engineering a product. Especially with regards to computationally intensive classification algorithms, we found that a simple, but well-calibrated solution, often significantly outperformed mathematically complex implementations. However, we were also conscious of the expenses associated with rewriting code that was under-engineered. We fell into a similar dilemma when deciding how much flexibility we should allow ourselves in deviating from our original proposal. Often times, our work would lead us to better solutions to the problems we faced, but sometimes these solutions would not fit into the framework that we had initially proposed. By keeping these factors in balance and making our project modular and our set of goals relatively focused, we believe we were able to create functional software.

In addition to these generalities, we learned to have several options when using an API to solve a component of the problem. In this project, we wanted to deal with the recognition algorithms and use existing software for feature extraction. However, we were often faced with problems in the functionality of 3Gears API or the Sphinx and Google Voice projects. We learned that having increased flexibility in being able to replace any part of the project without compromising our goals is always preferable. However, the physical costs associated with 3D sensors and limited availability of licensable feature extraction APIs made many of these choices for us. Nevertheless, the project was a highly educational experience in collaborative software prototyping and we hope that we will be able to use our developing skillset on similar projects in the future.

Reflections on Creativity and Imagination:

This project tested the bounds of our creativity and imagination in unexpected ways. We knew that gesture classification is a relatively young field, and expected to have to try many different approaches to get satisfactory results. However, we were also surprised with how effective recognition problems could be using simple innovations. For example, we knew of dynamic time warp and sliding window algorithms that could be used to classify gestures independent of when they were started or how quickly they were performed. But on a hunch, we decided to try segmentation and linearly scaling our testing samples to the size of our training samples. The fix was as simple as a clever constant multiplied by the counter in a for-loop, but the results were surprisingly accurate at solving our problem and significantly better than our initial attempts at sliding windows and dynamic time warp. As our project progressed, we found ourselves attempting many of these simple innovations and we greatly enjoyed the way in which they pushed the bounds of our creativity and imagination. We still believe that there are a plethora of new ways that 3D gesturing or even voice commands could be used to interact with computers and we are excited for the creative and imaginative possibilities that abound.

Potential Applications:

Although we used the video game, Little Fighter 2, to test our software, there are numerous potential applications for our work. One of our testing applications for our gestural interface extends a vector from the hand to the screen and attempts to place the mouse on that location. The result, codenamed "Minority Report Demo", allows us to use 3D gesturing to control the mouse. Pairing our voice commands and 3D gestures with actions, we believe that our product could be used as the primary interface for most computing tasks. Moreover, we believe that its benefits in ease-of-use and practicality would likely be greater for these applications than for the game that we demonstrated, and for many of these applications, error thresholds could be greatly widened without significant detriment to the user experience or usability of the product. In addition to allowing humans to interact with machines in unique and powerful ways, we believe that such an interface would be useful for disabled people or children. Our system would be a good starting point for an automated American Sign Language interpreter or an interface for blind or deaf people to use computers. It could also allow young children, who would lack the dexterity and size to touch-type or use a traditional keyboard or mouse, to begin using computers. We are greatly excited for these possibilities and believe that we were surely see them become a reality as computing power, sensors, and classification systems improve.

Acknowledgements:

This project was completed as part of CMSC23400 Mobile Computing (Winter 2013), taught by Professor Chien, with TA support by Tanakorn L. and Falcon Dai. We'd like to thank 3Gears for providing us an Academic License to use their software. Also thanks to Marti and Starsky Wong for making Little Fighter 2. Code is available on Github under the project name HenryVsRudolph.

References:

lmezain, Mahmoud, Ayoub Al-hamadi, and Bernd Michaelis. "A Hidden Markov Model-Based Isolated and Meaningful Hand Gesture Recognition." International Journal of Electrical and Electronic Engineering 3.3 (2009): 156-63. Print.

Jang, Roger. "12-2." Audio Signal Processing. Web. <http://neural.cs.nthu.edu.tw/jang/books/audiosignalprocessing/speechFeatureMfcc.asp?title=12-2%20MFCC>.

Kiran, Kranthi. Speech Recognition Using MFCC and Dynamic Time Warping. Indian Institute of Technology, Web. <http://xa.yimg.com/kq/groups/24321415/1523383180/name/Speech_Recognition_seminar.pdf>.

Lee, Hyeon-Kyu, and Jin H. Kim. "An HMM-Based Threshold Model Approach for Gesture Recognition." IEEE Transactions On Pattern Analysis And Machine Learning 21.10 (1999): 961-73. Print.

Shen, Dawei. Some Mathematics for HMM. MIT, Web. <http://courses.media.mit.edu/2010fall/mas622j/ProblemSets/ps4/tutorial.pdf>.